OIPE IAP80
FEB 2 4 2006
PATENT & TRADEMARK OFFICE

41260

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Application of : **BEER et al.**
:
Serial No.: 10/045,007 : Group Art Unit: 2193
:
Filed : January 15, 2002 : Examiner: Jason D. Mitchell
:
For : AUTOMATIC ABSTRACTION OF SOFTWARE
SOURCE CODE

Honorable Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

DECLARATION UNDER 37 CFR 1.131

Sir:

We, the undersigned, Ilan Beer and Cindy Eisner, hereby declare as follows:

1) We are the Applicants in the patent application identified above, and are the inventors of the subject matter described and claimed in claims 1-4, 6, 7, 9-16, 18, 20-28, 30, 31 and 33-36 therein.

2) Prior to July 15, 2000, we reduced our invention to practice, as described and claimed in the subject application, in Israel, a WTO country. We implemented the invention in the form of software program code in the C programming language. When compiled and run, this program performed the function of converting software source code into a finite-state model,

BEST AVAILABLE COPY

which was then verified using the RuleBase model checker produced by our employer, IBM Haifa Research Laboratory. We described the invention, as well as our experience in testing the invention on the garbage collection mechanism of the SMV model checker, in U.S. Provisional Patent Application 60/261,539, filed January 15, 2001, from which the present patent application claims priority.

3) As evidence of the reduction to practice of the present invention, we attach hereto in Exhibit A software source code that we used to implement the invention. This version of the code was frozen and archived in a TAR file on a date prior to July 15, 2000. A directory listing of the TAR file is attached hereto in Exhibit B, showing the date on which the source code files were archived. The dates that are blacked out in Exhibit B are prior to July 15, 2000. For brevity, only the code files main.c and new.c are included in Exhibit A. The remaining files are available upon request.

4) Generally speaking, the software code in Exhibit A performs the functions of processing source code to derive a set of next-state functions representing control flow of the source code, replacing the references to program variables in the source code with non-deterministic choices in the next-state functions, and restricting the next-state functions to produce a finite-state model of the control flow. As noted earlier, RuleBase was used to verify the finite-state model. The following table shows the correspondence between the elements of the method claims in the present patent application and elements of the material in Exhibit A:

US 10/045,007

Declaration under 37 C.F.R 1.131 by Beer et al.

Claim 1	Exhibit A
1. A computer-implemented method for verifying software source code that includes references to program variables	The method is carried out by the program main.c, together with the associated source files gram.y, scan.l, new.c, hash2.c listed in Exhibit B. The program variables of the software source code to be verified are identified in new_id() in new.c (page 22 in Exhibit A).
processing the source code to derive a set of next-state functions representing control flow of the source code	The next-state functions are derived from the source code by the output_pcl() routine in main.c (page 13 in Exhibit A).
replacing the references to the program variables in the source code with non-deterministic choices in the next-state functions	Each program variable is replaced by a non-deterministic choice, referred to as pcaux%s, in the output_pcl() routine in main.c. The replacement itself is carried out by statements starting "if (controlonly)".

restricting the next-state functions including the non-deterministic choices to produce a finite-state model of the control flow	The stack depth of the next-state functions is restricted to a certain maximum, referred to as maxstack, in the output_stack_and_stackp() routine in main.c (page 3 in Exhibit A), thus producing a finite-state model.
wherein replacing the references to the program variables comprises eliminating the references to the program variables from the next-state functions, so that the finite-state model is independent of data values of the program variables	The output_pc1() routine in main.c removes all references to the program variables (see "if (controlonly)"). Therefore, the finite-state model is independent of the data values of the program variables.
verifying the finite-state model to find an error in the source code	This function was carried out by RuleBase.
Claim 2	
2. A method according to claim 1, wherein processing the source code comprises extracting a program counter from the source code, and expressing the next-state functions in terms of the program counter.	The number_it() routine in main.c (page 12 in Exhibit A) extracts the program counter (referred to as pc). The next-state functions generated by the number_it() routine mentioned above use this counter, as can be seen in output_pc1().

Claim 3	
3. A method according to claim 2, wherein processing the source code further comprises expressing the next-state functions with reference to a stack pointer associated with a stack used in running the code	The stack pointer is referred to as stackp in the next-state functions, as can be seen in output_stack_and_stackp() in main.c (page 3 in Exhibit A). The stack is referred to as stack_%%(ii).
wherein replacing the program variables comprises eliminating all the references to the program variables from the next-state functions, leaving the next-state functions dependent on the program counter and on the stack pointer.	Since all the program variables were removed from the next-state functions in the output_pc1() routine, while the program counter and stack pointer were included in the next-state functions, as explained above, the next-state functions remain dependent on the program counter and stack pointer.
Claim 4	
4. A method according to claim 3, wherein restricting the next-state functions comprises limiting the stack pointer to a value no greater than a predetermined maximum.	As noted above, the stack depth of the next-state functions is restricted to a certain maximum, referred to as maxstack, in the output_stack_and_stackp() routine in main.c (page 3 in Exhibit A).

Claim 6	
6. A method according to claim 1, wherein processing the source code further comprises expressing the next-state functions with reference to a stack used in running the code, and wherein restricting the next-state functions comprises limiting the stack to a depth no greater than a predetermined maximum.	See comments above regarding claims 3 and 4, where similar limitations to these appear.
Claim 7	
7. A method according to claim 6, wherein expressing the next-state functions comprises expressing the next-state functions in terms of a stack pointer associated with the stack, and wherein limiting the stack comprises limiting the stack pointer to a value no greater than the predetermined maximum	See comments above regarding claim 4, in which the stack pointer is introduced.

wherein expressing the next-state functions in terms of the stack pointer comprises incrementing the stack pointer in response to a function call in the source code, up to the predetermined maximum, and decrementing the stack pointer when the function returns.	The stack pointer (referred to as stackp) is incremented and decremented in response to function calls and returns in the output_stack_and_stackp() routine in main.c (page 3 in Exhibit A).
Claim 9	
9. A method according to claim 1, wherein verifying the finite-state model comprises checking the finite-state model against a specification using a model checker.	This function is performed by the RuleBase model checker.
Claim 10	

10. A method according to claim 9, wherein restricting the next-state functions comprises automatically producing the model from the source code in a form suitable for processing by the model checker, without human intervention in deriving and restricting the next-state functions or in replacing the references.	The model is produced automatically by the code in Exhibit A. Specifically, the main() routine in main.c generates the final model for input to the model checker.
Claim 11	
11. A method according to claim 9, wherein checking the finite state model comprises checking the model against one or more formulas expressed in terms of temporal logic.	RuleBase accepts as input formulas expressed in terms of temporal logic and uses these formulas in checking finite state models.
Claim 12	
12. A method according to claim 9, wherein checking the finite state model comprises finding a counter-example indicative of the error.	When RuleBase determines that a specification property has been violated, it generates and outputs a counter-example.

5) Claims 13-16, 18, 20-28, 30, 31 and 33-36 recite

apparatus and a computer software product, with limitations similar to those of method claims 1-4, 6, 7 and 9-12. Based on the similarity of subject matter between the method, apparatus and software claims, it can similarly be demonstrated that we reduced to practice the entire invention recited in claims 13-16, 18, 20-28, 30, 31 and 33-36 prior to July 15, 2000.

6) The software code in Exhibit A was compiled and run prior to July 15, 2000, in order to test the garbage collection mechanism of the SMV program. (The results of this work were later described in the above-mentioned provisional patent application.) We attach hereto as Exhibit C several bug reports that were generated in the course of this testing, indicating specific faults in SMV that were found through the use of our invention. The dates that are blacked out on these bug reports are prior to July 15, 2000. The bug reports demonstrate that our invention was successfully used for its intended purpose.

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

US 10/045,007

Declaration under 37 C.F.R 1.131 by Beer et al.

Cindy Ei

Ilan Beer
Citizen of Israel
3 Petel Street
Haifa 34556
Israel

Date:
9 Feb 2006

Cindy Eisner
Citizen of U.S. and Israel
6 Heharuv Street
Zichron Yaakov 30900
Israel

Date:
Feb 9, 2006

EXHIBIT A

MAIN.C

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <math.h>
#define LINESIZE 1024
#define MAXSUFFIX 7 /* max suffix is ".ranges" */
#define MAXVARS 100
#define MAXFUNS 100
#define MAXSTACK 5
#define MAXRETURNS 500
#define MAXSCOPES 100
#define MAXINDICES 100
#include <node.h>
#include <types.h>
#include <hash2.h>

ste_ptr scopes[MAXSCOPES];
int nscopes=0;
ste_ptr scope = NIL;
hash2_ptr symbol_table;
char input_filename[NAMELENGTH];
char *temp_filename;
char c_output_filename[NAMELENGTH];
char edl_output_filename[NAMELENGTH];
char range_filename[NAMELENGTH];
FILE *c_output_file;
FILE *edl_output_file;
FILE *range_file;
void number_it_if_function_not_topfunction(hash2_item_ptr v);
void spit_it_out_if_function(hash2_item_ptr v);
static int pc = 0;
int starttopfunction=-1, endtopfunction=-1;
int pcnocall=0;
int nreturns=0;
int returns[MAXRETURNS];
char *topfunction;
int nindices=0;
node_ptr indices[MAXINDICES];
int controlonly = 0;
int ii = 0;
int cpc=0;
int maxcases=2;
int doingtopfunction;


char *format_str( char *format, ... )
{
   static char str[5000];
   va_list pvar;

   str[sizeof(str)-1] = 0;
   va_start(pvar, format);
   vsprintf( str, format, pvar );
   va_end(pvar);
   if ( str[sizeof(str)-1] != 0 )
      catastrophe( "Internal error: format_str string is too long" );
   return str;
}


double log2 (x)
double x;
{
return(log(x) / log(2));
}

main(argc,argv)
int argc;
char **argv;
{
  char *myname;

  myname = *(argv++);
  if (!strcmp(*argv,"-control")) {
    controlonly = 1;
    argv++;
    argc--;
  }
  if (argc != 3) {
    fprintf(stderr,"invocation:  %s [-control] cfile function \n",myname);
    exit(1);
  }
  open_inputs(*(argv++));
  topfunction = *argv;
  open_outputs(*(argv++));

  symbol_table = create_hash2();

  yyparse();
  for_all_hash2_items(symbol_table, number_it_if_function_not_topfunction);
  number_topfunction(); /* number topfunction last purely for debugging
purposes */
  for_all_hash2_items(symbol_table, spit_it_out_if_function);
  spit_out_vars();
  read_range_info();

  output_startrule();
  output_pc();

  output_nextpcnocall();
  output_stack_and_stackp();
  output_vars_and_calls();
  output_endrule();
  if (!controlonly)
    output_indices();

  fclose(c_output_file);
  fclose(edl_output_file);
}

read_range_info()
{
  char line[LINESIZE];
  char *maxnstring, *scopename, *name;
  int maxn;
  ste_ptr st;

  if (!range_file)
    return;
  while (fgets(line,LINESIZE,range_file)) {
    name = strtok(line," \t\n");
    scopename = strtok(NULL," \t\n");
    maxnstring = strtok(NULL," \t\n");
    if(maxnstring==NULL) {
      maxnstring = scopename;
      scopename = NULL;
    }
    if (!name)
      continue;
    st =
find_hash2(symbol_table,name,scopename?find_hash2(symbol_table,scopename,NU
LL):NULL);
    if (!st) {
      fprintf(stderr,"cannot find variable %s
%s%s%s\n",name,scopename?"(":"",scopename,scopename?"(":"");
    }
    else {
      sscanf(maxnstring,"%d",&maxn);
      st -> maxn = maxn;
    }
  }
}
output_startrule()
{
  fprintf(edl_output_file,"rule %s {\n",topfunction);
  fprintf(edl_output_file,"test_pins pcint,returntowhereint;\n");
}
output_endrule()
{
  fprintf(edl_output_file,"#include \"%s.formula\"\n",topfunction);
  fprintf(edl_output_file,"}\n");
}

output_stack_and_stackp()
{
  int i;

  fprintf(edl_output_file,"var stackp: 0.. %d;\n",MAXSTACK+1);
  fprintf(edl_output_file,"%%for ii in 0..%d %%do\n",MAXSTACK);
  fprintf(edl_output_file,"var stack_%%{ii}(0..%d): boolean;\n",cpc);
  fprintf(edl_output_file,"%%end\n");
  fprintf(edl_output_file,"assign init(stackp) := 0;\n");
  fprintf(edl_output_file,"       next(stackp) := case\n");
  fprintf(edl_output_file,"                         somerealcall: if
stackp=%d then %d else stackp + 1 endif;\n",MAXSTACK+1,MAXSTACK+1);
  fprintf(edl_output_file,"                         somereturn: if stackp=0
then 0 else stackp - 1 endif;\n");
  fprintf(edl_output_file,"                         else: stackp;\n");
  fprintf(edl_output_file,"       esac;\n");
  fprintf(edl_output_file,"invar stackp != %d;\n",MAXSTACK+1);
  fprintf(stderr,"warning:  stack limited to depth of %d\n",MAXSTACK);
  for (i=0;i<=MAXSTACK;i++) {
    fprintf(edl_output_file,"assign next(stack_%d(0..%d)) :=
case\n",i,cpc);
    fprintf(edl_output_file,"                           %d != stackp:
stack_%d(0..%d);\n",i,i,cpc);
    fprintf(edl_output_file,"                           else:
nextpcnocall(0..%d);\n",cpc);
    fprintf(edl_output_file,"       esac;\n");
  }
  fprintf(edl_output_file,"define stackpminus1 := if stackp = 0 then 0 else
stackp - 1 endif;\n");
  fprintf(edl_output_file,"define returntowhere(0..%d) := case\n",cpc);
  for (i=0;i<=MAXSTACK;i++) {
     fprintf(edl_output_file,"stackpminus1=%d:stack_%d(0..%d);\n",i,i,cpc);
  }
  fprintf(edl_output_file,"esac;\n");
}

spit_out_one_var_for_list(hash2_item_ptr v)
{
  ste_ptr p;
  p = (ste_ptr) v -> p;
  if (!(p->isfunction | p->istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(c_output_file,"  %s",p->name);
    if (p -> scope)
      fprintf(c_output_file,"(%s)",p->scope->name);
    if (p -> isarray)
      fprintf(c_output_file,"[%d]",p->arraybound);
    fprintf(c_output_file,"\n");
  }
}
spit_out_one_var(ste_ptr p)
{
  if (!(p->isfunction | p->istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(edl_output_file,"%s",p->name);
    if (p -> scope)
      fprintf(edl_output_file,"_%s",p->scope->name);
  }
}

spit_out_vars()
{
  int i;

  fprintf(c_output_file,"/* vars: \n");
  for_all_hash2_items(symbol_table, spit_out_one_var_for_list);
  fprintf(c_output_file,"*/\n");
}

output_one_var_for_this_parse_tree(hash2_item_ptr v, ste_ptr varp)
{
  ste_ptr p;

  p = (ste_ptr) v -> p;
  output_one_var(p -> parse_tree, varp, 0);
}

output_one_if_var(hash2_item_ptr v)
{
  ste_ptr p;
  char *varname;

  p = (ste_ptr) v -> p;
  varname = v -> name;
  if (!(p -> isfunction | p->istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(edl_output_file,"var ");
    spit_out_one_var(p);
    if (p -> isarray)
      fprintf(edl_output_file,"(0..%d)",p -> arraybound);
    if (p -> maxn == 1)
      fprintf(edl_output_file,": boolean;\n");
    else
      fprintf(edl_output_file,": 0..%d;\n",p -> maxn);
    if (p -> nassignments) {
      if (p -> isarray)
        fprintf(edl_output_file,"%%for __ijk in 0..%d do\n",p ->
arraybound);
      fprintf(edl_output_file,"assign next(");
      spit_out_one_var(p);
      if (p -> isarray) {
        fprintf(edl_output_file,"(__ijk)");
      }
      fprintf(edl_output_file,") :=case\n");
      for_all_hash2_items_1moreparam(symbol_table,
output_one_var_for_this_parse_tree,p);
      fprintf(edl_output_file,"else: ");
      spit_out_one_var(p);
      if (p -> isarray) {
        fprintf(edl_output_file,"(__ijk)");
      }
      fprintf(edl_output_file,";\n");
      fprintf(edl_output_file,"esac;\n");
      if (p -> isarray)
        fprintf(edl_output_file,"%%end\n");
    }
  }
}
output_use_if_var(hash2_item_ptr v)
{
  ste_ptr p;
  char *varname;
  int j;

  p = (ste_ptr) v -> p;
  varname = v -> name;
  if (!(p -> isfunction | p -> istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(edl_output_file,"define useof_");
    spit_out_one_var(p);
    fprintf(edl_output_file," := 0");
    if (p -> nuses)
      for (j=0;j<p -> nuses;j++) {
        fprintf(edl_output_file,"|(pc(0..%d)=%d)",cpc,p -> uses[j]);
        if (j && !(j%10))
          fprintf(edl_output_file,"\n");
      }
    fprintf(edl_output_file,";\n");
  }
}
output_useastopparam_if_var(hash2_item_ptr v)
{
  ste_ptr p;
  char *varname;
  int j;

  p = (ste_ptr) v -> p;
  varname = v -> name;
  if (!(p -> isfunction | p -> istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(edl_output_file,"define useastopparam_");
    spit_out_one_var(p);
    fprintf(edl_output_file," := 0");
    if (p -> ntopparamuses)
      for (j=0;j<p -> ntopparamuses;j++) {
        fprintf(edl_output_file,"|(pc(0..%d)=%d)",cpc,p ->
topparamuses[j]);
        if (j && !(j%10))
          fprintf(edl_output_file,"\n");
      }
    fprintf(edl_output_file,";\n");
  }
}
output_assign_if_var(hash2_item_ptr v)
{
  ste_ptr p;
  char *varname;
  int j;
  p = (ste_ptr) v -> p;
  varname = v -> name;
  if (!(p -> isfunction | p -> istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(edl_output_file,"define assignto_");
    spit_out_one_var(p);
    fprintf(edl_output_file," := 0");
    if (p -> nassignments)
      for (j=0;j<p -> nassignments;j++) {
        fprintf(edl_output_file,"|(pc(0..%d)=%d)",cpc,p -> assignments[j]);
        if (j && !(j%10))
          fprintf(edl_output_file,"\n");
      }
    fprintf(edl_output_file,";\n");
  }
}
output_orassignto_if_var(hash2_item_ptr v)
{
  ste_ptr p;
  char *varname;

  p = (ste_ptr) v -> p;
  varname = v -> name;
  if (!(p -> isfunction | p -> istype | p->isstructorunionfield | p-
>isstructorunion)) {
    fprintf(edl_output_file,"|assignto_");
    spit_out_one_var(p);
    if (ii++ && !(ii%10))
      fprintf(edl_output_file,"\n");
  }
}
output_orcallto_if_fun_withbody(hash2_item_ptr v)
{
  ste_ptr p;
  char *funname;

  p = (ste_ptr) v -> p;
  funname = v -> name;
  if (p -> isfunction && p -> parse_tree) {
    fprintf(edl_output_file,"|callto_%s",funname);
    if (ii++ && !(ii%10))
      fprintf(edl_output_file,"\n");
  }
}
number_topfunction()
{
  ste_ptr p;

  p = find_hash2(symbol_table,topfunction,NIL);
  if (!p) {
    catastrophe("couldn't find topfunction");
  }
  starttopfunction = p -> startpc = pc;
  number_it(p -> parse_tree);
  endtopfunction = p -> endpc = pc;
}
void number_it_if_function_not_topfunction(hash2_item_ptr v)
{
  ste_ptr p;
  char *funname;

  p = (ste_ptr) v -> p;
  funname = v -> name;
  if (p -> isfunction && p->parse_tree && strcmp(funname,topfunction)) {
    p -> startpc = pc;
    number_it(p -> parse_tree);
    p -> endpc = pc;
    note_return(pc-1);
  }
}
void spit_it_out_if_function(hash2_item_ptr v)
{
  ste_ptr p;
  char *funname;

  p = (ste_ptr) v -> p;
  funname = v -> name;
  if (p -> isfunction && p -> parse_tree) {
    fprintf(c_output_file,"%s()\n",p -> name);
    if (!p -> parse_tree -> next)
      fprintf(c_output_file,"{\n",p -> name);
    spit_it_out(p -> parse_tree,0,0);
    if (!p -> parse_tree -> next)
      fprintf(c_output_file,"}\n",p -> name);
  }
}

output_callto_if_fun(hash2_item_ptr v)
{
  ste_ptr p;
  char *funname;
  int j;

  p = (ste_ptr) v -> p;
  funname = v -> name;
  if (p -> isfunction) {
    fprintf(edl_output_file,"define callto_%s := 0",funname);
    if (p -> ncalls)
      for (j=0;j<p -> ncalls;j++) {
        fprintf(edl_output_file,"|(pc(0..%d)=%d)",cpc,p -> calls[j]);
        if (j && !(j%10))
          fprintf(edl_output_file,"\n");
      }
    fprintf(edl_output_file,";\n");
  }
}

output_indices()
{
  int i;
  for (i=0;i<nindices;i++) {
    fprintf(edl_output_file,"define __index%d := ",i);
    print_edlexpr(indices[i]);
    fprintf(edl_output_file,";\n");
  }
}
output_vars_and_calls()
{
  int i,j;

  if (controlonly) {
    for_all_hash2_items(symbol_table, output_use_if_var);
    for_all_hash2_items(symbol_table, output_useastopparam_if_var);
    for_all_hash2_items(symbol_table, output_assign_if_var);
    fprintf(edl_output_file,"define someassign := 0");
    for_all_hash2_items(symbol_table, output_orassignto_if_var);
    fprintf(edl_output_file,";\n");
    for_all_hash2_items(symbol_table, output_callto_if_fun);
    fprintf(edl_output_file,"define somerealcall := 0");
    ii = 0;
    for_all_hash2_items(symbol_table, output_orcallto_if_fun_withbody);
    fprintf(edl_output_file,";\n");
  }
    fprintf(edl_output_file,"define somereturn := 0");
    for (i=0;i<nreturns;i++) {
      fprintf(edl_output_file,"|pc(0..%d)=%d",cpc,returns[i]);
      if (i && !(i%10))
        fprintf(edl_output_file,"\n");
    }
    fprintf(edl_output_file,";\n");
  if (!controlonly)
    for_all_hash2_items(symbol_table, output_one_if_var);
}

spit_it_out(sp,isieq,isipp)
node_ptr sp;
int isieq,isipp;
{
  int compound;
  node_ptr left;

  if (sp -> next && sp -> stype != CASE_st)
    compound = 1;
  else
    compound = 0;
  if (compound && !isipp) fprintf(c_output_file,"{\n");
  while (sp) {
    if (!isieq && !isipp)
      fprintf(c_output_file,"/* %d */ ",sp -> pc);
    switch(sp -> stype) {
      case IF_st:
           fprintf(c_output_file,"if (");
           print_expr(sp -> conde,0,sp -> pc,0,1);
           fprintf(c_output_file,")\n");
           spit_it_out(sp -> thens,0,0);
           if (sp -> elses) {
             fprintf(c_output_file,"else \n");
             spit_it_out(sp -> elses,0,0);
           }
           break;
      case CASE_st:
           fprintf(c_output_file,"case ");
           print_expr(sp -> label,0,sp -> pc,0,1);
           fprintf(c_output_file,":");
           spit_it_out(sp -> doit,0,0);
           break;
      case SWITCH_st:
           fprintf(c_output_file,"switch (");
           print_expr(sp -> switche,0,sp -> pc,0,1);
           fprintf(c_output_file,") {\n");
           spit_it_out(sp -> caselist,0,0);
           fprintf(c_output_file,"}\n");
           break;
      case FOR_st:
           fprintf(c_output_file,"for (");
           spit_it_out(sp -> ieq,1,0);
           print_expr(sp -> ilt,0,sp -> pc,0,1);
           fprintf(c_output_file,";\n/* %d */",sp -> pc + 1);
           spit_it_out(sp -> ipp,0,1);
           fprintf(c_output_file,")\n");
           spit_it_out(sp -> dowhat,0,0);
           break;
      case WHILE_st:
           fprintf(c_output_file,"while (");
           print_expr(sp -> conde,0,sp -> pc,0,1);
           fprintf(c_output_file,")\n");
           spit_it_out(sp -> dowhat,0,0);
           break;
      case ASSIGNMENT_st:
           if (sp -> left -> etype == TIMES_ex)
             left = sp -> left -> left;
           else
             left = sp -> left;
           if (left -> etype != IDENTIFIER_ex)
             catastrophe("don't know what to do with assignment");
           note_assignment(left -> stinfo -> name,left -> stinfo ->
scope,isieq?sp -> pc -1 :sp -> pc);
           fprintf(c_output_file,"%s",left -> stinfo -> name);
           if (left -> index) {
             fprintf(c_output_file,"[");
             print_expr(left -> index,0,pc,0,0);
             fprintf(c_output_file,"]");
           }
           fprintf(c_output_file," = ");
           print_expr(sp -> right,0,sp -> pc,0,1);
           if (isieq)
             fprintf(c_output_file,";");
           else if (isipp) {
             if (sp -> next)
               fprintf(c_output_file,",");
           }
           else
             fprintf(c_output_file,";\n");
           break;
      case RETURN_st:
           fprintf(c_output_file,"return;\n");
           break;
      case BREAK_st:
           fprintf(c_output_file,"break;\n");
           break;
      case CONTINUE_st:
           fprintf(c_output_file,"continue;\n");
           break;
      case EMPTY_st:
           fprintf(c_output_file,";\n");
           break;
      case PREINC_ex:
      case INC_ex:
           note_assignment(sp -> left -> stinfo -> name, sp -> left ->
stinfo -> scope,sp -> pc);
           if (sp -> stype == PREINC_ex)
             fprintf(c_output_file,"++");
           fprintf(c_output_file,"%s",sp -> left -> stinfo -> name);
           if (sp -> stype == INC_ex)
             fprintf(c_output_file,"++");
           if (isipp) {
             if (sp -> next)
               fprintf(c_output_file,",");
           }
           else
             fprintf(c_output_file,";\n");
           break;
      case PREDEC_ex:
      case DEC_ex:
           note_assignment(sp -> left -> stinfo -> name, sp -> left ->
stinfo -> scope,sp -> pc);
           if (sp -> stype == PREDEC_ex)
             fprintf(c_output_file,"--");
           fprintf(c_output_file,"%s",sp -> left -> stinfo -> name);
           if (sp -> stype == DEC_ex)
             fprintf(c_output_file,"--");
           if (isipp) {
             if (sp -> next)
               fprintf(c_output_file,",");
           }
           else
             fprintf(c_output_file,";\n");
           break;
      case FCALL_ex:
           print_expr(sp -> fun,1,sp -> pc,0,1);
           fprintf(c_output_file,"(");
           print_expr(sp -> argulist,0,sp -> pc,1,1);
           fprintf(c_output_file,");\n");
           break;
      default:  catastrophe(format_str("ERROR:  unknown statement type
%i",sp -> stype));
    }
    sp = sp -> next;
  }
  if (compound && !isipp) fprintf(c_output_file,"}\n");
}

print_expr(ep,isfunction,pc,isargulist,topcall)
node_ptr ep;
int isfunction,pc,isargulist,topcall;
{
while (ep) {
  switch (ep -> etype) {
    case ASSIGNMENT_st:
      spit_it_out(ep,0,1);
      break;
    case FCALL_ex:
      print_expr(ep -> fun,1,pc,0,0);
      fprintf(c_output_file,"(");
      print_expr(ep -> argulist,0,pc,topcall?1:0,0);
      fprintf(c_output_file,")");
      break;
    case IDENTIFIER_ex:
      if (isfunction) {
        note_call(ep -> stinfo -> name, pc);
      }
      else {
        note_use(ep -> stinfo -> name, ep -> stinfo -> scope, pc);
        if (isargulist)
          note_useastopparam(ep -> stinfo -> name, ep -> stinfo -> scope,
pc);
      }
      fprintf(c_output_file,"%s",ep -> stinfo -> name);
      if (ep -> index) {
        fprintf(c_output_file,"[");
        print_expr(ep -> index,0,pc,0,0);
        fprintf(c_output_file,"]");
      }
      break;
    case STRING_ex:
      fprintf(c_output_file,"%s",ep -> stext);
      break;
    case DEC_ex:
      print_expr(ep -> left,0,pc,0,0);
      fprintf(c_output_file,"--");
      break;
    case PREDEC_ex:
      fprintf(c_output_file,"--");
      print_expr(ep -> left,0,pc,0,0);
  break;
case INC_ex:
  print_expr(ep -> left,0,pc,0,0);
  fprintf(c_output_file,"++");
  break;
case PREINC_ex:
  fprintf(c_output_file,"++");
  print_expr(ep -> left,0,pc,0,0);
  break;
case CONSTANT_ex:
  switch (ep -> format) {
    case 'd': fprintf(c_output_file,"%d",(int) ep -> val); break;
    case 'o': fprintf(c_output_file,"0%o",(int) ep -> val); break;
    case 'x': fprintf(c_output_file,"0x%x",(int) ep -> val); break;
    case 'f': if (controlonly)
                fprintf(c_output_file,"%f",ep -> val);
              else
                catastrophe("no support for reals");
              break;
    default:  catastrophe("unknown format");
  }
  break;
case NOT_ex:
  fprintf(c_output_file,"!");
  print_expr(ep -> left,0,pc,0,0);
  break;
case DEFAULT_ex:
  fprintf(c_output_file,"default");
  break;
case LT_ex:
case GT_ex:
case LE_ex:
case GE_ex:
case EQ_ex:
case NE_ex:
case BITAND_ex:
case XOR_ex:
case BITOR_ex:
case AND_ex:
case OR_ex:
case PLUS_ex:
case MINUS_ex:
case TIMES_ex:
case DIV_ex:
case SHIFTLEFT_ex:
case SHIFTRIGHT_ex:
case MOD_ex:
  if (ep -> right) /* else unary! */
    print_expr(ep -> left,0,pc,0,0);
  switch (ep -> etype) {
   case LT_ex: fprintf(c_output_file," < "); break;
   case GT_ex: fprintf(c_output_file," > "); break;
   case LE_ex: fprintf(c_output_file," <= "); break;
   case GE_ex: fprintf(c_output_file," >= "); break;
   case EQ_ex: fprintf(c_output_file," == "); break;
   case NE_ex: fprintf(c_output_file," != "); break;
   case BITAND_ex: fprintf(c_output_file," & "); break;
   case XOR_ex: fprintf(c_output_file," ^ "); break;
   case BITOR_ex: fprintf(c_output_file," | "); break;
   case AND_ex: fprintf(c_output_file," && "); break;
   case OR_ex: fprintf(c_output_file," || "); break;
   case PLUS_ex: fprintf(c_output_file," + "); break;
   case MINUS_ex: fprintf(c_output_file," - "); break;
   case TIMES_ex: fprintf(c_output_file," * "); break;
   case DIV_ex: fprintf(c_output_file," / "); break;
```

```
       case SHIFTLEFT_ex: fprintf(c_output_file," << "); break;
       case SHIFTRIGHT_ex: fprintf(c_output_file," >> "); break;
       case MOD_ex: fprintf(c_output_file," % "); break;
      default:  catastrophe(format_str("ERROR:  unknown expression type
%i",ep -> etype));
      }
      print_expr(ep -> right,0,pc,0,0);
      break;
    default:  catastrophe(format_str("ERROR:  unknown expression type
%i",ep -> etype));
  }
  if (ep -> next)
    fprintf(c_output_file,",");
  ep = ep -> next;
}
}

number_it(sp)
node_ptr sp;
{
  node_ptr ipp;

  while (sp) {
    switch(sp -> stype) {
      case IF_st:
           sp -> pc = pc++;
           number_it(sp -> thens);
           if (sp -> elses)
             number_it(sp -> elses);
           break;
      case CASE_st:
           sp -> pc = pc; /* sic */
           number_it(sp -> doit);
           break;
      case SWITCH_st:
           sp -> pc = pc++;
           number_it(sp -> caselist);
           break;
      case FOR_st:
           sp -> pc = pc++;
           number_it(sp -> ieq);
           number_it(sp -> dowhat);
           ipp = sp -> ipp;
           while (ipp) {
             ipp -> pc = pc;
             ipp = ipp -> next;
           }
           pc++;
           break;
      case WHILE_st:
           sp -> pc = pc++;
           number_it(sp -> dowhat);
           break;
      case INC_ex:
      case DEC_ex:
      case ASSIGNMENT_st:
           sp -> pc = pc++;
           break;
      case RETURN_st:
           note_return(pc);
      case FCALL_ex:
      case BREAK_st:
      case CONTINUE_st:
      case EMPTY_st:
           sp -> pc = pc++;
           break;
      default:  catastrophe(format_str("ERROR:  unknown statement type
%i",sp -> stype));
    }
    sp = sp -> next;
  }
}

open_inputs(char *filename)
{
  extern int yylineno;
  extern FILE *yyin;
  char *cpp_command;
  int cpp_wait_code;

  if (strlen(filename) > (NAMELENGTH-MAXSUFFIX-1)) {
    catastrophe(format_str("filename too long: %s\n",filename));
  }
  strcpy(input_filename,filename);
  strcat(input_filename,".c");

    temp_filename = tmpnam(NULL);
    cpp_command = format_str (
        "cpp -I. < %s > %s", input_filename, temp_filename);

    cpp_wait_code = system (cpp_command);
    if (cpp_wait_code) {
        fprintf (stderr, "cpp failed with code %d\n", cpp_wait_code);
        unlink (temp_filename);
        catastrophe( "preprocessing failed (maybe /tmp is full)\n" );
    }

  if(!(yyin = fopen(temp_filename,"r")))
     catastrophe(format_str("cannot open %s for input\n", temp_filename ));
  yylineno = 1;

  if (controlonly)
    return;

  strcpy(range_filename,filename);
  strcat(range_filename,".ranges");
  if (!(range_file = fopen(range_filename,"r")))
     fprintf(stderr,"warning:  cannot find range file %s\n", range_filename
);
}

open_outputs(char *filename)
{
  strcpy(edl_output_filename,filename);
  strcat(edl_output_filename,".edl");
  if(!(edl_output_file = fopen(edl_output_filename,"w")))
     catastrophe(format_str("cannot open %s for output",
edl_output_filename ));
  strcpy(c_output_filename,filename);
  strcat(c_output_filename,".cout");
  if(!(c_output_file = fopen(c_output_filename,"w")))
     catastrophe(format_str("cannot open %s for output", c_output_filename
));
}

void output_pc1_if_function(hash2_item_ptr v)
{
  ste_ptr p;
  char *funname;
  p = (ste_ptr) v -> p;
  funname = v -> name;
  if (p -> isfunction) {
    if (strcmp(funname,topfunction))
      doingtopfunction=0;
    else
      doingtopfunction=1;
    output_pc1(p -> parse_tree,p -> endpc, p -> endpc, p -> parse_tree ->
pc);
  }
}
output_pc()
{
  cpc = (int) ceil(log2((double) pc));
  fprintf(edl_output_file,"define pcint := bvtoi(pc(0..%d));\n",cpc);
  fprintf(edl_output_file,"define returntowhereint :=
bvtoi(returntowhere(0..%d));\n",cpc);
  fprintf(edl_output_file,"var pc(0..%d): boolean;\n", cpc);
  fprintf(edl_output_file,"assign init(pc(0..%d)) :=
%d;\n",cpc,starttopfunction);
  fprintf(edl_output_file,"assign next(pc(0..%d)) := case\n",cpc);
  fprintf(edl_output_file,"somereturn: returntowhere(0..%d);\n",cpc);
  for_all_hash2_items(symbol_table, output_pc1_if_function);
  fprintf(edl_output_file,"pc(0..%d)=%d:
%d;\n",cpc,endtopfunction,endtopfunction);
  fprintf(edl_output_file,"esac;\n");
  fprintf(edl_output_file,"define maxpc := %d;\n",endtopfunction);
  fprintf(edl_output_file,"var pcaux: 0..%d;\n", maxcases-1);
}
output_nextpcnocall()
{
  pcnocall=1;
  fprintf(edl_output_file,"define nextpcnocall(0..%d) := case\n",cpc);
  for_all_hash2_items(symbol_table, output_pc1_if_function);
  fprintf(edl_output_file,"pc(0..%d)=%d:
%d;\n",cpc,endtopfunction,endtopfunction);
  fprintf(edl_output_file,"esac;\n");
}

output_pc1(sp,nextfather,breaktowhere,continuewhere)
node_ptr sp;
int nextfather,breaktowhere,continuewhere;
{
  ste_ptr fun;

  while (sp) {
    if (sp -> stype != CASE_st)
      fprintf(edl_output_file,"pc(0..%d)=%d:",cpc,sp -> pc);
    switch(sp -> stype) {
      case IF_st:
           if (controlonly)
             fprintf(edl_output_file,"if pcaux=0 then %d else %d
endif;\n",sp -> thens -> pc, sp -> elses? sp -> elses -> pc: sp -> next? sp
-> next -> pc : nextfather);
           else {
             fprintf(edl_output_file,"if (");
             print_edlexpr(sp -> conde);
             fprintf(edl_output_file,") then %d else %d endif;\n",sp ->
thens -> pc, sp -> elses? sp -> elses -> pc: sp -> next? sp -> next -> pc :
nextfather);
           }
           output_pc1(sp -> thens,sp -> next? sp -> next -> pc:
nextfather,breaktowhere,continuewhere);
           output_pc1(sp -> elses,sp -> next? sp -> next -> pc:
nextfather,breaktowhere,continuewhere);
```

```
           break;
      case CASE_st:
           output_pc1(sp -> doit, sp -> next? sp -> next -> pc:
nextfather,breaktowhere,continuewhere);
           break;
      case SWITCH_st:
           if (controlonly) {
             node_ptr caselist;
             int ncases;

             fprintf(edl_output_file,"case ");
             caselist = sp -> caselist;
             ncases=0;
             while (caselist) {
               if (caselist -> stype == CASE_st) {
                 if (caselist -> next && caselist -> next -> stype ==
CASE_st)
                   fprintf(edl_output_file,"pcaux = %d",ncases++);
                 else
                   fprintf(edl_output_file,"else");
                 fprintf(edl_output_file, ": %d; ",caselist -> pc);
               }
               caselist = caselist -> next;
             }
             fprintf(edl_output_file,"esac;\n");
             maxcases = (ncases > maxcases)?ncases:maxcases;
           }
           else {
             node_ptr caselist;

             fprintf(edl_output_file,"case ");
             caselist = sp -> caselist;
             while (caselist) {
               if (caselist -> stype == CASE_st) {
                 print_edlexpr(sp -> switche);
                 fprintf(edl_output_file, " = ");
                 print_edlexpr(caselist -> label);
                 fprintf(edl_output_file, ": %d; ",caselist -> pc);
               }
               caselist = caselist -> next;
             }
             fprintf(edl_output_file,"esac;\n");
           }
           output_pc1(sp -> caselist, sp -> next? sp -> next -> pc:
nextfather, sp -> next? sp -> next -> pc: nextfather,continuewhere);
           break;
      case FOR_st:
           fprintf(edl_output_file,"%d;\n",sp -> pc + 1);
           fprintf(edl_output_file,"pc(0..%d)=%d:",cpc,sp -> pc + 1);
           if (controlonly)
             fprintf(edl_output_file,"if pcaux=0 then %d else %d
endif;\n",sp -> dowhat -> pc, sp -> next? sp -> next -> pc: nextfather);
           else {
             fprintf(edl_output_file,"if (");
             print_edlexpr(sp -> ilt);
             fprintf(edl_output_file,") then %d else %d endif;\n",sp ->
dowhat -> pc, sp -> next? sp -> next -> pc: nextfather);
           }
           sp -> endpc = (sp -> next? sp -> next -> pc: nextfather) - 1;
           output_pc1(sp -> dowhat, sp -> endpc, sp -> next? sp -> next ->
pc: nextfather,sp -> pc + 1);
           fprintf(edl_output_file,"pc(0..%d)=%d: %d;\n",cpc,sp -> endpc,sp
-> pc + 1);
           break;
      case WHILE_st:
           if (controlonly)
             fprintf(edl_output_file,"if pcaux=0 then %d else %d
endif;\n",sp -> dowhat -> pc, sp -> next? sp -> next -> pc: nextfather);
           else {
             fprintf(edl_output_file,"if (");
             print_edlexpr(sp -> conde);
             fprintf(edl_output_file,") then %d else %d endif;\n",sp ->
dowhat -> pc, sp -> next? sp -> next -> pc: nextfather);
           }
           output_pc1(sp -> dowhat, sp -> pc, sp -> next? sp -> next -> pc:
nextfather,sp -> pc);
           break;
      case FCALL_ex:
           if (pcnocall)
             fprintf(edl_output_file,"%d;\n",sp -> next? sp -> next -> pc:
nextfather);
           else {
             fun = sp -> fun -> stinfo;
             if (fun -> parse_tree)
               fprintf(edl_output_file,"%d;\n",fun -> parse_tree -> pc);
             else
               fprintf(edl_output_file,"%d;\n",sp -> next? sp -> next ->
pc: nextfather);
           }
           break;
      case ASSIGNMENT_st:
           if (sp -> right -> etype == FCALL_ex) {
             if (pcnocall)
               fprintf(edl_output_file,"%d;\n",sp -> next? sp -> next ->
pc: nextfather);
             else {
               fun = sp -> right -> fun -> stinfo;
               if (fun -> parse_tree)
                 fprintf(edl_output_file,"%d;\n",fun -> parse_tree -> pc);
               else
                 fprintf(edl_output_file,"%d;\n",sp -> next? sp -> next ->
pc: nextfather);
             }
           }
           else
             fprintf(edl_output_file,"%d;\n",sp -> next? sp -> next -> pc:
nextfather);
           break;
      case INC_ex:
      case DEC_ex:
      case EMPTY_st:
           fprintf(edl_output_file,"%d;\n",sp -> next? sp -> next -> pc:
nextfather);
           break;
      case RETURN_st:
           if (doingtopfunction)
             fprintf(edl_output_file,"itobv(maxpc);\n");
           else
             fprintf(edl_output_file,"returntowhere(0..%d);\n",cpc);
           break;
      case CONTINUE_st:
           fprintf(edl_output_file,"%d;\n",continuewhere);
           break;
      case BREAK_st:
           fprintf(edl_output_file,"%d;\n",breaktowhere);
           break;
      default:  catastrophe(format_str("ERROR:  unknown statement type
%i",sp -> stype));
    }
    sp = sp -> next;
```

```
  }
}

print_edlexpr(ep)
node_ptr ep;
{
  static indexcount = 0;
  if (controlonly)
    catastrophe("shouldn't have gotten to here\n");
  switch (ep -> etype) {
    case FCALL_ex:
      fprintf(edl_output_file,"{0,1}");
      break;
    case IDENTIFIER_ex:
      spit_out_one_var(ep -> stinfo);
      if (ep -> index) {
        fprintf(edl_output_file,"(");
        if (ep -> index -> etype != CONSTANT_ex) {
          fprintf(edl_output_file,"__index%d",nindices);
          note_index(nindices++,ep -> index);
          if (!ep -> stinfo -> isarray)
            catastrophe("bad array");
          fprintf(edl_output_file,": 0..%d",ep -> stinfo -> arraybound);
        }
        else
          print_edlexpr(ep -> index);
        fprintf(edl_output_file,")");
      }
      break;
    case STRING_ex:
      fprintf(edl_output_file,"%s",ep -> stext);
      break;
    case CONSTANT_ex:
      switch (ep -> format) {
        case 'd': fprintf(edl_output_file,"%d",(int) ep -> val); break;
        case 'o': fprintf(edl_output_file,"0%o",(int) ep -> val); break;
        case 'x': fprintf(edl_output_file,"0x%x",(int) ep -> val); break;
        case 'f': catastrophe("no support for reals");
        default:  catastrophe("unknown format");
      }
      break;
    case NOT_ex:
      fprintf(edl_output_file,"!");
      print_edlexpr(ep -> left);
      break;
    case DEFAULT_ex:
      fprintf(edl_output_file,"else");
      break;
    case LT_ex:
    case GT_ex:
    case LE_ex:
    case GE_ex:
    case EQ_ex:
    case NE_ex:
    case BITAND_ex:
    case XOR_ex:
    case BITOR_ex:
    case AND_ex:
    case OR_ex:
    case PLUS_ex:
    case MINUS_ex:
    case TIMES_ex:
    case DIV_ex:
    case SHIFTLEFT_ex:
    case SHIFTRIGHT_ex:
```

```
    case MOD_ex:
      print_edlexpr(ep -> left);
      switch (ep -> etype) {
        case LT_ex: fprintf(edl_output_file," < "); break;
        case GT_ex: fprintf(edl_output_file," > "); break;
        case LE_ex: fprintf(edl_output_file," <= "); break;
        case GE_ex: fprintf(edl_output_file," >= "); break;
        case EQ_ex: fprintf(edl_output_file," == "); break;
        case NE_ex: fprintf(edl_output_file," != "); break;
        case BITAND_ex: fprintf(edl_output_file," & "); break;
        case XOR_ex: fprintf(edl_output_file," ^ "); break;
        case BITOR_ex: fprintf(edl_output_file," | "); break;
        case AND_ex: fprintf(edl_output_file," && "); break;
        case OR_ex: fprintf(edl_output_file," || "); break;
        case PLUS_ex: fprintf(edl_output_file," + "); break;
        case MINUS_ex: fprintf(edl_output_file," - "); break;
        case TIMES_ex: fprintf(edl_output_file," * "); break;
        case DIV_ex: fprintf(edl_output_file," / "); break;
        case SHIFTLEFT_ex: fprintf(edl_output_file," << "); break;
        case SHIFTRIGHT_ex: fprintf(edl_output_file," >> "); break;
        case MOD_ex: fprintf(edl_output_file," % "); break;
        default:  catastrophe(format_str("ERROR:  unknown expression type
%i",ep -> etype));
      }
      print_edlexpr(ep -> right);
      break;
    default:  catastrophe(format_str("ERROR:  unknown expression type
%i",ep -> etype));
  }
}

note_use(varname,scope,pc)
char *varname;
int pc;
ste_ptr scope;
{
  ste_ptr st;

  st = find_hash2(symbol_table,varname,scope);
  if (st -> nuses >= MAXUSES)
    catastrophe(format_str("too many uses of variable %s\n",varname));
  st -> uses[st -> nuses++] = pc;
}
note_useastopparam(varname,scope,pc)
char *varname;
int pc;
ste_ptr scope;
{
  ste_ptr st;

  st = find_hash2(symbol_table,varname,scope);
  if (st -> ntopparamuses >= MAXUSES)
    catastrophe(format_str("too many uses of variable %s as
parameter\n",varname));
  st -> topparamuses[st -> ntopparamuses++] = pc;
}
note_call(funname,pc)
char *funname;
int pc;
{
  ste_ptr st;

  st = find_hash2(symbol_table,funname,NIL);
  if (st -> ncalls >= MAXUSES)
    catastrophe(format_str("too many calls to function %s\n",funname));
  st -> calls[st -> ncalls++] = pc;
}
note_return(pc)
int pc;
{
  if (nreturns >= MAXRETURNS)
    catastrophe(format_str("too many returns \n"));
  returns[nreturns++] = pc;
}
note_index(n,expr)
int n;
node_ptr expr;
{
  if (n >= MAXINDICES)
    catastrophe(format_str("too many indexed variables \n"));
  indices[n] = expr;
}
note_assignment(varname,scope,pc)
char *varname;
int pc;
ste_ptr scope;
{
  ste_ptr st;

  st = find_hash2(symbol_table,varname,scope);
  if (st -> nassignments >= MAXASSIGNMENTS)
    catastrophe(format_str("too many assignments to variable
%s\n",varname));
  st -> assignments[st -> nassignments++] = pc;
}

output_one_var(sp,varp,isieq)
node_ptr sp;
ste_ptr varp;
int isieq;
{
  int i;

  while (sp) {
    switch(sp -> stype) {
      case IF_st:
           output_one_var(sp -> thens,varp,0);
           if (sp -> elses)
             output_one_var(sp -> elses,varp,0);
           break;
      case CASE_st:
           output_one_var(sp -> doit,varp,0);
           break;
      case SWITCH_st:
           output_one_var(sp -> caselist,varp,0);
           break;
      case FOR_st:
           output_one_var(sp -> ieq,varp,1);
           output_one_var(sp -> dowhat,varp,0);
           output_one_var(sp -> ipp, varp, 0);
           break;
      case WHILE_st:
           output_one_var(sp -> dowhat,varp,0);
           break;
      case ASSIGNMENT_st:
           if (varp == sp -> left -> stinfo) {
             fprintf(edl_output_file,"pc(0..%d)=%d: ",cpc,isieq?sp -> pc -1
:sp -> pc);
             if (!sp -> left -> index) {
               print_edlexpr(sp -> right);
               fprintf(edl_output_file,";\n");
             }
             else {
               fprintf(edl_output_file,"if __ijk=");
               print_edlexpr(sp -> left -> index);
               fprintf(edl_output_file," then ");
               print_edlexpr(sp -> right);
               fprintf(edl_output_file," else ");
               spit_out_one_var(sp -> left -> stinfo);
               fprintf(edl_output_file,"(__ijk) endif;\n");
             }
           }
           break;
      case FCALL_ex:
      case BREAK_st:
      case CONTINUE_st:
      case EMPTY_st:
      case RETURN_st:
           break;
      case INC_ex:
           if (varp == sp -> left -> stinfo) {
             fprintf(edl_output_file,"pc(0..%d)=%d: if ",cpc,sp -> pc);
             print_edlexpr(sp -> left);
             fprintf(edl_output_file," + 1 > %d then {",varp -> maxn);
             for (i=0;i<varp -> maxn;i++)
               fprintf(edl_output_file,"%d,",i);
             fprintf(edl_output_file,"%d",varp -> maxn);
             fprintf(edl_output_file,"} else ");
             print_edlexpr(sp -> left);
             fprintf(edl_output_file," + 1 endif;\n");
           }
           break;
      case DEC_ex:
           if (varp == sp -> left -> stinfo) {
             fprintf(edl_output_file,"pc(0..%d)=%d: if ",cpc,sp -> pc);
             print_edlexpr(sp -> left);
             fprintf(edl_output_file," - 1 < 0 then {");
             for (i=0;i<varp -> maxn;i++)
               fprintf(edl_output_file,"%d,",i);
             fprintf(edl_output_file,"%d",varp -> maxn);
             fprintf(edl_output_file,"} else ");
             print_edlexpr(sp -> left);
             fprintf(edl_output_file," - 1 endif;\n");
           }
           break;
      default:  catastrophe(format_str("ERROR:  unknown statement type
%i",sp -> stype));
    }
    sp = sp -> next;
  }
}
catastrophe(char *s)
{
  extern int yylineno;
  extern char *current_input_filename;

  fprintf(stderr,"fatal error on line %d file %s:
%s\n",yylineno,current_input_filename,s);
  exit(1);
}
node_ptr clean_out_empties(st)
node_ptr st;
{
  node_ptr ret, last_non_empty;
  while(st && (st -> stype == EMPTY_st)) {
    if (st -> next)
      st -> next -> last = st -> last;
    st = st -> next;
  }
  ret = st;
  last_non_empty = ret;
  while (st && st -> next) {
    if (st -> next -> stype == EMPTY_st) {
      st -> next = st -> next -> next;
    }
    else
      last_non_empty = st -> next;
    st = st -> next;
  }
  if (ret)
    ret -> last = last_non_empty;
  return(ret);
}
push_scope(f)
ste_ptr f;
{
  if (nscopes > MAXSCOPES)
    catastrophe("scope overflow");
  scopes[nscopes++] = scope;
  scope = f;
}
pop_scope()
{
  scope = scopes[--nscopes];
  if (nscopes < 0)
    catastrophe("scope underflow");
}
```

NEW.C

```
#include <stdio.h>
#include <node.h>
#include <types.h>
#include <hash2.h>

extern hash2_ptr symbol_table;
extern ste_ptr scope;
node_ptr new_assign();
node_ptr new_compound();


ste_ptr new_stentry(char *name)
{
  ste_ptr ret;

  ret = (ste_ptr) malloc(sizeof(struct stentry));
  ret -> name = name;
  ret -> type = NIL;
  ret -> scope = NIL;
  ret -> ispointer = 0;
  ret -> isfunction = 0;
  ret -> isarray = 0;
  ret -> maxn = 1;
  ret -> parse_tree = NIL;
  ret -> istype = 0;
  ret -> istypedef = 0;
  ret -> isextern = 0;
  ret -> isstructorunionfield = 0;
  ret -> isstructorunion = 0;
  ret -> typep = NIL;
  ret -> nuses = 0;
  ret -> ntopparamuses = 0;
  ret -> nassignments = 0;
  ret -> ncalls = 0;
  return(ret);
}
ste_ptr new_type(name,type)
{
  ste_ptr ret;

  ret = find_hash2(symbol_table,name,NIL);
  if (ret && !ret -> istype)
    catastrophe("found type as non-type in symbol table");
  if (ret)
    return(ret);
  ret = new_stentry(NIL);
  ret -> name = (char *)strcpy((char *)malloc(strlen(name)+1),name);
  ret -> type = type;
  ret -> istype = 1;
  insert_hash2(symbol_table, name, NIL, ret);
  return(ret);
}

node_ptr new_id(name, isdef, scope)
char *name;
int isdef;
ste_ptr scope;
{
  node_ptr ret;
  ste_ptr st;
  int scope_level,i;
  char temp[NAMELENGTH],tempname[NAMELENGTH],tempchar;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = IDENTIFIER_ex;
  st = find_hash2(symbol_table,name,scope);
  if (!st && !isdef) {
    if (scope) {
      tempchar = scope->name[strlen(scope->name)-1];
      if (tempchar >= '0' && tempchar <= '9') {
        scope_level = tempchar - '0';
        for (i=scope_level-1;i>=0;i--) {
          strcpy(temp,scope->name);
          temp[strlen(scope->name)-1] = 0;
          sprintf(tempname,"%s%d",temp,i);
          st =
find_hash2(symbol_table,name,find_hash2(symbol_table,tempname,NIL));
          if (st)
            break;
        }
      }
    }
    if (!st) {
      strcpy(temp,scope->name);
      temp[strlen(scope->name)-1] = 0;
      st =
find_hash2(symbol_table,name,find_hash2(symbol_table,&temp[2],NIL));
    }
    if (!st)
      st = find_hash2(symbol_table,name,NIL);
    if (!st) {
      /* if we didn't find it and it isn't a def, assume it is a global */
      ret -> stinfo = new_stentry((char *)strcpy((char
*)malloc(strlen(name)+1),name));
      insert_hash2(symbol_table, name, NIL, ret -> stinfo);
    }
    else
      ret -> stinfo = st;
  }
  else if (st) {
    if (st -> isfunction && !st -> parse_tree) {
      ret -> stinfo = st;
    }
    else if (isdef && !st -> isstructorunion && !st -> isstructorunionfield
&& !st -> isextern)
      catastrophe(format_str("fatal error: symbol already defined:
%s\n",name));
    else ret -> stinfo = st;
  }
  else {
    ret -> stinfo = new_stentry((char *)strcpy((char
*)malloc(strlen(name)+1),name));
    ret -> stinfo -> scope = scope;
    insert_hash2(symbol_table, name, scope, ret -> stinfo);
  }
  return(ret);
}
node_ptr new_sizeofcall(argulist)
node_ptr argulist;
{
  node_ptr ret,fun;
  ste_ptr st;

  fun = new_id("sizeof", 0, NIL);
  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = FCALL_ex;
  ret -> fun = fun;
  st = fun -> stinfo;
  if (st)
    st -> isfunction = 1;
  else
    catastrophe("couldn't find function");
  ret -> argulist = argulist;
  return(ret);
}

node_ptr rep_node_nonext(st)
node_ptr st;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  memcpy (ret, st, sizeof(struct node));
  ret -> next = NULL;
  ret -> last = NULL;
  return(ret);
}

node_ptr new_fcall(dummyfun,argulist)
node_ptr dummyfun,argulist;
{
  node_ptr ret;
  ste_ptr st;
  node_ptr fun;
  extern int yylineno;
  node_ptr temps,temp,temp2;
  static int tempcount=0;
  node_ptr newargulist;
  char *text,*name;

  newargulist = NULL;
  temps = NULL;
  while (argulist) {
    if (argulist -> etype == FCALL_ex) {
      if (temps && argulist -> temps)
        temps = new_compound(temps,argulist -> temps);
      else if (argulist -> temps)
        temps = argulist -> temps;
      argulist -> temps = NULL;
      text = format_str("__tempforcall%d",tempcount++);
      name = (char *)strcpy((char *)malloc(strlen(text)+1),text);
      temp = new_id(name,0,NULL);
      temp2 = new_assign(temp,rep_node_nonext(argulist));
      if (temps)
        temps = new_compound(temps,temp2);
      else
        temps = temp2;
    }
    else
      temp = rep_node_nonext(argulist);
    if (newargulist)
      newargulist = new_compound(newargulist,temp);
    else
      newargulist = temp;
    argulist = argulist -> next;
  }

  if (dummyfun -> etype != IDENTIFIER_ex) {
    fprintf(stderr,"warning line %d:  not yet supported:  call by pointer
to function.  assuming call is atomic\n",yylineno);
    fun = new_id("__dummyfun",0,NIL);
    fun -> stinfo -> isfunction = 1;
  }
  else if (dummyfun -> stinfo -> scope != NIL) {
    catastrophe("this should have been fixed\n");
/*
    fun = new_id(dummyfun -> stinfo -> name, 0, NIL);
    fun -> stinfo -> isfunction = 1;
    remove_hash2(symbol_table,dummyfun -> stinfo -> name, dummyfun ->
stinfo -> scope);
*/
  }
  else
    fun = dummyfun;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = FCALL_ex;
  ret -> fun = fun;
  st = fun -> stinfo;
  if (st)
    st -> isfunction = 1;
  else
```

```
    catastrophe("couldn't find function");
  ret -> argulist = newargulist;
  ret -> temps = temps;
  return(ret);
}
node_ptr new_string_literal(text)
char *text;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = STRING_ex;
  ret -> stext = (char *)strcpy((char *)malloc(strlen(text)+1),text);
  return(ret);
}
node_ptr new_tempid(text)
char *text;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = TEMPID_ex;
  ret -> stext = (char *)strcpy((char *)malloc(strlen(text)+1),text);
  return(ret);
}

node_ptr new_default()
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = DEFAULT_ex;
  return(ret);
}

node_ptr new_inc(incwhat)
node_ptr incwhat;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = INC_ex;
  ret -> left = incwhat;
  return(ret);
}
node_ptr new_dec(decwhat)
node_ptr decwhat;
{
  node_ptr ret;
  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = DEC_ex;
  ret -> left = decwhat;
  return(ret);
}
node_ptr new_preinc(incwhat)
node_ptr incwhat;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = PREINC_ex;
  ret -> left = incwhat;
  return(ret);
}
node_ptr new_predec(decwhat)
node_ptr decwhat;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = PREDEC_ex;
  ret -> left = decwhat;
  return(ret);
}

node_ptr new_constant(i,format)
double i;
char format;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = CONSTANT_ex;
  ret -> val = i;
  ret -> format = format;
  return(ret);
}

node_ptr new_binary(op,left,right)
int op;
node_ptr left, right;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> etype = op;
  ret -> left = left;
  ret -> right = right;
  if (left && left -> temps && right && right -> temps)
    ret -> temps = new_compound(left -> temps, right -> temps);
  else if (left && left -> temps)
    ret -> temps = left -> temps;
  else if (right)
    ret -> temps = right -> temps;
  else
    ret -> temps = NULL;
  if (left)
    left -> temps = NIL;
  if (right)
    right -> temps = NIL;
  return(ret);
}

node_ptr new_if(conde,thens,elses)
node_ptr conde;
node_ptr thens,elses;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = IF_st;
  ret -> conde = conde;
  ret -> thens = thens;
  ret -> elses = elses;
  return(ret);
}

node_ptr new_switch(switche,caselist)
node_ptr switche;
node_ptr caselist;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = SWITCH_st;
  ret -> switche = switche;
  ret -> caselist = caselist;
  return(ret);
}

node_ptr new_while(conde,dowhat)
node_ptr conde;
node_ptr dowhat;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = WHILE_st;
  ret -> conde = conde;
  ret -> dowhat = dowhat;
  return(ret);
}

node_ptr new_for(ieq,ilt,ipp,dowhat)
node_ptr ilt,ipp;
node_ptr ieq,dowhat;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = FOR_st;
  ret -> ieq = ieq;
  ret -> ilt = ilt;
  ret -> ipp = ipp;
  ret -> dowhat = dowhat;
  return(ret);
}

node_ptr new_case(label,doit)
node_ptr label;
node_ptr doit;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = CASE_st;
  ret -> label = label;
  ret -> doit = doit;
  return(ret);
}

node_ptr new_assign(left,right)
node_ptr left,right;
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = ASSIGNMENT_st;
  ret -> left = left;
  ret -> right = right;
  return(ret);
}

node_ptr new_break()
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = BREAK_st;
  return(ret);
}
node_ptr new_continue()
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = CONTINUE_st;
  return(ret);
}
node_ptr new_empty()
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = EMPTY_st;
  return(ret);
}
node_ptr new_return()
{
  node_ptr ret;

  ret = (node_ptr) malloc(sizeof(struct node));
  ret -> temps = NIL;
  ret -> index = NIL;
  ret -> next = NIL;
  ret -> last = NIL;
  ret -> stype = RETURN_st;
  return(ret);
}

node_ptr new_compound(list,one)
node_ptr list, one;
{
  node_ptr ret;

  ret = list;
  if (ret -> last)
    ret -> last -> next = one;
  else
    ret -> next = one;
  if (one -> last)
    ret -> last = one -> last;
  else
    ret -> last = one;
  return(ret);
}
```

EXHIBIT B

2/6/2006 11:16 AM **Page 1**

C:\DOCUME~1\DANIEL~1\LOCALS~1\Temp\c2edl.[redacted].tar

Name	**Modified**	**Size**	**Ratio**	**Packed**	**Path**
gram.y	[redacted] 12:49 PM	23,155	0%	23,155	afs\haifa\home\eis...\freeze_[redacted]
hash2.c	[redacted] 12:49 PM	4,520	0%	4,520	afs\haifa\home\eis...\freeze_[redacted]
hash2.h	[redacted] 12:49 PM	2,573	0%	2,573	afs\haifa\home\eis...\freeze_[redacted]
main.c	[redacted] 12:49 PM	38,931	0%	38,931	afs\haifa\home\eis...\freeze_[redacted]
Makefile	[redacted] 12:49 PM	479	0%	479	afs\haifa\home\eis...\freeze_[redacted]
new.c	[redacted] 12:49 PM	11,666	0%	11,666	afs\haifa\home\eis...\freeze_[redacted]
node.h	[redacted] 12:49 PM	2,007	0%	2,007	afs\haifa\home\eis...\freeze_[redacted]
scan.l	[redacted] 12:49 PM	6,988	0%	6,988	afs\haifa\home\eis...\freeze_[redacted]
types.h	[redacted] 12:49 PM	1,213	0%	1,213	afs\haifa\home\eis...\freeze_[redacted]
y.output	[redacted] 12:49 PM	110,824	0%	110,824	afs\haifa\home\eis...\freeze_[redacted]
y.tab.h	[redacted] 12:49 PM	1,422	0%	1,422	afs\haifa\home\eis...\freeze_[redacted]
11 file(s)		**203,778**	**0%**	**203,778**	

EXHIBIT C

Issue1709.detail

*** Created by (eisner) on [redacted] at 12:18.
*** Owner: (vanna)

in for_all_ffs (macro), there is a missing release_bdd() for the last step of the for loop! (found by smv on smv).

see trace at:

/afs/haifa.ibm.com/proj6/fprojects9/software/hrl_smv/traces/001

*** Commentary by (eisner) on [redacted] at 10:18.

found by smv on smv

*** Answered by (vanna) on [redacted] at 14:43.

there is release at the end of macro
no need to fix

*** Closed by (eisner) on 13 Dec 2000 at 15:17.

ok.

Issue1710.detail

*** Created by (eisner) on [redacted] at 10:17.
*** Owner: (vanna)

there is a missing save_bdd() for variable cand in function check_eq() of file reduce.c

see:

$PROJ9/software/hrl_smv/traces/002

(found by smv on smv)

*** Commentary by (eisner) on [redacted] at 14:55.

same problem exists for variable tmp_out in this function (same for loop).
see:

$PROJ9/software/hrl_smv/traces/006

*** Answered by (vanna) on [redacted] at 14:44.

fixed in reduce.c rev1.52

*** Closed by (eisner) on 13 Dec 2000 at 15:18.

ok.

Issue 1711.detail

*** Created by (eisner) on [redacted] at 11:04.
*** Owner: (vanna)

there is a missing release_bdd() for variable atom in function cone_of_inf.

see:

/afs/haifa.ibm.com/proj6/fprojects9/software/hrl_smv/traces/003

*** Answered by (vanna) on [redacted] at 14:44.

fixed in reduce.c rev1.52

*** Closed by (eisner) on 13 Dec 2000 at 15:17.

ok.

Issue 1717.detail

*** Created by (eisner) on [redacted] at 13:27.
*** Owner: (vanna)

missing save_bdd() for variable c in function check_const() in file reduce.c. see:

/afs/haifa.ibm.com/proj6/fprojects9/software/hrl_smv/traces/004

*** Answered by (vanna) on [redacted] at 14:45.

fixed in reduce.c rev1.52

*** Closed by (eisner) on 13 Dec 2000 at 15:17.

ok.

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

☒ **BLACK BORDERS**

□ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**

□ **FADED TEXT OR DRAWING**

□ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**

□ **SKEWED/SLANTED IMAGES**

□ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**

□ **GRAY SCALE DOCUMENTS**

□ **LINES OR MARKS ON ORIGINAL DOCUMENT**

□ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**

□ **OTHER:** ______________________________

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.